P. Bresciani

The Challenge of Integrating Knowledge Representation and Data bases

Informatica 20 (4) 443-454, December 1996.

This paper appears in *Informatica 20 (4) 443–454*, December 1996.

The Challenge of Integrating Knowledge Representation and Databases

Paolo Bresciani
Knowledge Representation and Reasoning group
Istituto per la Ricerca Scientifica e Tecnologica,
via Sommarive 18, I-38050 Povo (Trento), Italy.
bresciani@irst.itc.it

December 1996

Abstract

Two different aspects of data management are addressed by Knowledge Representation (KR) and Databases (DB): the semantic organization of data and powerful reasoning services by KR, and their efficient management and access by DB. It is recently emerging that experiences from both KR and DB should profitably cross-fertilize each other, and a great interest is rising about this topic.

In particular, among several ways to approach knowledge representation, Description Logics (DL) are gaining, in the last years, a privileged place. In this paper, after briefly showing the importance of an integrated view of description logics and databases, our approach to this topic is presented. Our technique allows uniform access – by means of a DL-based query language – to information distributed over knowledge bases and databases. The separately existing retrieving functions of description logics management systems and of database management systems are integrated, in our extended paradigm, in order to allow, via a query language grounded on a DL-based schema knowledge, uniformly formulating and answering queries and, thus, uniform retrieval from mixed knowledge/data bases.

1 Introduction

Traditional Information Science applications that need the storage and management of large amounts of data rely on database tools, while AI applications that need the managing and reasoning about more structured data – like for example expert systems applications, natural language interfaces [12], decision support systems – rely on knowledge representation systems.

In fact, the main difference between knowledge representation and database systems is that the latter are oriented to the efficient management of large amounts of data, while the former seek to give a more structured representation of the universe of discourse in which data are placed. More specifically, in some knowledge representation systems the application domain is described by means of a collection of complex terms – or *concepts* – that are placed into a taxonomy. The capability of *classifying* concepts to form taxonomies, accordingly with well defined semantics, is given by an appropriate calculus, whose first goal is to provide a *subsumption* algorithm. Concept languages together with semantically grounded subsumption calculi are called description logics.[1] Databases, instead, are suited to manage data efficiently, with little concern about their dimension, but the formalism for organizing them in a structured way is quite absent, as well as the capability to infer new information from that already existing. Thus, two different aspects of data management are addressed by Description Logics Management Systems (DLMS) and by Database Management Systems (DBMS): the semantic organization of data by DLMS, and their efficient management and access by DBMS.

The importance of knowledge representation in general has been regarded as fundamental for the construction of good Intelligent Information Systems [17] for more than ten years (see, e.g., [31]), but only recently the theoretical foundations of a uniform approach to description logics and databases have been established [18].

From another point of view, KR-based applications and, more generally, AI-based applications can be widely enhanced by AI/DB interfaces [28, 26].

More in general, it is recently emerging that experiences from both KR and DB should profitably cross-fertilize each other, and a wide interest is rising on this topic (see [2, 3, 4]).

In this paper, description logics are considered as a privileged knowledge representation formalism, and it is proposed to deal with both description logics and databases together, using them in an integrated way, to manage different kinds of information. Of course, a uniform way to retrieve information from mixed knowledge/data bases is needed. Thus, it is shown how this task can be performed in a way completely transparent to the user. To this end, a semantically sound linking between the DL knowledge bases and databases is needed. Within the technique here presented, this is achieved by *tightly coupling* DLMS and DBMS [8].

In section 2 a very brief and informal introduction to description logics is given. Then, in section 3, some application fields of DL and DB, that can get advantages by using a uniform approach, are presented. Section 4, the main part of the paper, describes our technique for uniformly accessing – by means of a DL-based query language – information distributed over knowledge bases (KB) and databases. Some conclusive notes are conveyed in section 5.

2 Description Logics

Description logics are, essentially, variable-free concise reformulations of decidable restricted parts of First Order Logic (FOL). The syntax of DL is aimed to allow to define *concepts* (unary relation symbols), *roles* (binary relation symbols), and *individuals* (constants). FOL atomic sentences about constants can be expressed in DL by saying that an individual is an *instance* of a concept, or that a role holds

[1]Description logics are the most recent formalization of a family of knowledge representation languages and systems, known with several but almost equivalent names, as, e.g., Conceptual Languages, Terminological Logics, KL-ONE-like Systems [10], etc.

Construct	FOL Semantics
(`NOT` C)	$\neg F_C(\gamma)$
(`AND` C D)	$F_C(\gamma) \wedge F_D(\gamma)$
(`OR` C D)	$F_C(\gamma) \vee F_D(\gamma)$
(`ALL` R C)	$\forall x . F_R(\gamma, x) \Rightarrow F_C(x)$
(`SOME` R C)	$\exists x . F_R(\gamma, x) \wedge F_C(x)$
(`FILLED-BY` R a)	$F_R(\gamma, a)$
$\vdots$	$\vdots$
(`INVERSE` R)	$F_R(\beta, \alpha)$
(`COMPOSE` R Q)	$\exists x . F_R(\alpha, x) \wedge F_Q(x, \beta)$
(`AND-ROLE` R Q)	$F_R(\alpha, \beta) \wedge F_Q(\alpha, \beta)$
$\vdots$	$\vdots$

Table 1: FOL transformational semantics for a sample DL.

between two individuals. In different DL slightly different languages are used, with different levels of expressiveness. In all of them [21] complex concepts (and in some also complex roles) can be built starting from the atomic ones using different sets of operators to form *term descriptions*. Typical concept-forming operators are `AND`, `OR`, `NOT`, `ALL`, `SOME`, `FILLED-BY`; typical role-forming operators are `INVERSE`, `COMPOSE`, `AND-ROLE`. In the following, an intuitive LISP-like syntax for a generic DL will be used.

Description logics semantics can be briefly given by mapping DL expressions into FOL formulæ [18], as well as by a direct set-theoretic interpretation. Of course, these semantics are equivalent. We briefly give, here, the FOL transformational semantics: an atomic concept `A`, an atomic role `P`, and an individual a are respectively mapped – or *interpreted* – to the FOL open atomic formulæ $A(\gamma)$, $P(\alpha, \beta)$, and to the individual constant a. In table 1 for some DL complex concepts C the corresponding FOL open formulæ (with one free variable) $F_C(\gamma)$ are recursively given; similarly, for some DL complex roles R the corresponding FOL open formulæ (with two free variables) $F_R(\alpha, \beta)$ are recursively given.

For example, and very intuitively, the concept:

```
(AND
  Person
  (FILLED-BY has-sex FEMALE)
  (SOME has-children Person)
  (ALL has-children
        (AND
          Person
          (FILLED-BY has-sex MALE))))
```

denotes the class of mothers with only male children.[2] In every DLMS new atomic

[2]Of course, this holds if we assume to give to the concept `Person`, to the roles `has-sex` and `has-children`, and to the individuals `FEMALE` and `MALE`, the “natural meaning” their identifiers have in English.

concepts can be *defined* by giving names to concept expressions. In this case, of course, their semantics are those of their defining expressions. Also, *primitive* definitions can be formulated: a primitive definition gives only necessary but not sufficient conditions for an individual to belong to the primitively defined concept; that is, it cannot be inferred that an individual belong to a primitively defined concept unless it is explicitly asserted that it belongs to it or to a more specific concept in the taxonomy.[3] In other words, the semantics of a primitively defined concept is only partially given by the semantics of the primitively defining expression, and can be completed with any arbitrary restriction. A primitively defined concept (role) is simply called a *primitive concept* (*primitive role*).

For example, the previous concept expression could be the definition of `Mother--with-only-male-children`. Also the concepts `Mother` and `Woman` could be defined, as follows:

```
Mother ≐ (AND
            Woman
            (SOME has-children Person))

Woman ≐ (AND
           Person
           (FILLED-BY has-sex FEMALE)).
```

For each interpretation that is given to the atomic terms `Person`, `has-sex`, `has--children` (bound to no definition), and to the individuals `FEMALE`, and `MALE`, it is always the case that the class `Mother` includes the class `Mother-with-only-male--children`, as can be correctly checked by any DLMS whose language comprises the operators used in the example. It is said that `Mother` *subsumes* `Mother-with-only-male-children`.[4] The operation of organizing terms in a taxonomy based on subsumption is called *classification*.

Except for trivial cases, classification cannot be automatically performed in the entity relationship models of databases, even when concerning object oriented databases. On this respect, as it will be shown below, description logics can be useful in databases applications. On the other hand, DLMS are less efficient than DBMS in managing data about individual entities.

Typically, a DLMS contains not only a term definition module (TBox) – where concepts (and roles) definitions like those shown above are managed and classified – but also an assertional module (ABox) – where *assertions* about individuals are stated. Thus, relatively to the previous example, the following can be *asserted* in the ABox:

```
Person(SUE)
has-sex(SUE,FEMALE)
Person(TOM)
has-sex(TOM,MALE)
has-children(SUE,TOM)
```

[3]On the contrary, of course, an individual inherits all the properties of the concept it belongs to.

[4]According to the given FOL transformational semantics of DL, it can be said that a concept C subsumes a concept D iff $\forall x. F_D(x) \Rightarrow F_C(x)$.

that means that the individual `SUE` is an instance of the concept `Person`, the role `has-sex` holds between the two individuals `SUE` and `FEMALE`, and so on. From this, `Mother(SUE)` and – under the so called *closed world assumption* (CWA)[5] – `Mother-with-only-male-children(SUE)` follow.[6] This kind of inference is called *realization.*

An important tradeoff of DLMS is between the expressiveness of the description language characterizing the TBox and the inefficiency in managing large amounts of data in the ABox, even when the knowledge is organized in simple form based on primitive concepts, and, therefore, the realization is completely a-priori given. By contrast, DBMS are suited to manage data efficiently, with little concern about their amount, but their formalism for organizing data in a structured way is quite absent, as is the capability to infer new information from the existing one. The technique proposed in section 4 has been developed as a tool to overcome these DLMS and DBMS limitations.

3 Integrated use of DL and DB

Here, some application fields of DL and DB, where integrated approaches like the one mentioned above can be useful, are briefly listed. Far from being exhaustive, this short list can well be regarded as a sufficient motivation for going on in facing the challenge of integrating DL and DB.

3.1 Using DB for improving DL applications

DL applications can get benefits by using DB technology, for example in the following aspects.

- *Large ABox management.*
  As mentioned, the usual DLMS lack efficiency in managing large amount of data in the ABox. Our experience [12] suggests that, in realistic applications, knowledge bases not only can be complex, but can also involve a large number of individuals, difficult – when not impossible – to manage with the existing DLMS ABoxes. A large portion of data about them can be managed better by a DBMS.
- *Integrated KB and DB.*
  As noted in [8], KB based on DL are often used in applications where they need access to large amounts of data stored in already existing databases. Also in this case the approach proposed in section 4 is fruitful. A practical application of this can be found in the system TAMIC [5].
- *Knowledge Acquisition.*
  As observed in [13, 12] the task of acquiring knowledge for a real knowledge based application often includes a great amount of raw data collecting. For

[5]The CWA states that there are no other instances of a concept (or pairs of a role) except those provable to be [29]. The CWA represent, indeed, a crucial difference between databases semantics (that always assume it) and description logics semantics (that typically do not assume it), and is not yet fully addressed by the research here described.

[6]Because, for the CWA, her only child is `TOM`.

this subtask, instead of using ABox user interfaces, it can be more adequate to use databases, and their more robust front-ends. This is easily realizable when the paradigm proposed in the present paper is adopted.

3.2 Using DL for improving DB applications

DL can be useful in several fields of DB. Here, only the following few are listed.

- *Data Archaeology.*
  In *data archaeology* the main task is the search and extraction of previously ignored knowledge from several and possibly large databases, by means of an interactive and iterative process of analysis and refinement [11]. In this case, the use of intelligent front-ends based on DL can be helpful [20]. The use of friendly and powerful interfaces is even more important when data are widely and easily available to several kinds of users, but scarcely inter-organized and spread over several sites as in the case of the WWW [23]. In this case our architecture – that is not limited to databases access, but can be extended to more generic data collecting engines (see [13]) – can be used as a building block for defining DL mediated access systems.
- *Distributed and Heterogeneous Databases.*
  In this case the problem is to plan the optimal access to many, distributed, and eventually heterogeneous databases [22, 24]. DL is crucial on both the site descriptions meta-level, as noted in [24], and on the data level, as noted in [22]. Our paradigm can be used as a basic uniform access tool to the proposed DL/DB architecture.
- *Query Optimization.*
  This topic is worth of mention, even if the less related with our present work. Several query optimization techniques can be based on DL, both for a pure semantic organization of queries [18, 6], and as an aid for the pre-evaluation of the cardinality of queries in partially indexed databases [30].

Many other research trends on how description logics can be used for solving typical database problems are currently being explored. Two short surveys about them can be found in [15, 16].

4 DL access to DB

In the present section it is shown how assertional knowledge of DLMS and data of DBMS can be uniformly accessed from the DLMS in a way completely transparent to the user. This calls for a semantically sound linking between the DL knowledge base and the databases. It can be obtained by *coupling* DLMS and DBMS [8]: primitive concepts and relations in a KB are made to correspond respectively to unary and binary tables in a DB. In [8] two possible way to couple DLMS and DBMS are proposed:

- *loose coupling*, that requires a pre-loading of the data from the DB into the KB;
- *tight coupling*, that implements a *on demand* access to the DB;

but in the system presented in [20, 8] only the loose coupling paradigm is implemented.

Our approach, instead, is based on tight coupling of DLMS with DBMS, allowing the following advantages:

- complex compound queries (other than simply asking for the instances of a concept) can be done; for instance, conjunctive queries involving concepts and roles like $\mathtt{C}(x) \wedge \mathtt{R}(x, y) \wedge \mathtt{D}(y)$ can be done;
- no memory space is wasted in the DLMS in order to keep descriptions of DB data;
- answers are given on the basis of the current state of the KB and the DB;
- no periodical updating of the KB with new or modified data from the DB is needed.

4.1 DBox as an Extension of the ABox

The basic idea of our approach is to extend the traditional DLMS ABox with an external source of extensional data, called *DBox*.[7] By means of it, it is possible to couple the standard TBox/ABox architecture with one or more, possibly heterogeneous and distributed, databases, so that the user can make queries to this extended system without any concern on which DB or the KB have to be accessed, and the system is able to answer them uniformly.

In the following, the notation of [27] is adopted: a collection of term definitions – concepts and roles in a TBox – is called a *terminology* $\mathcal{T}$,[8] and a collection of assertions about individuals – in a ABox – is called a *world description* $\mathcal{W}$.[9] Moreover, the set of data expressed in a DBox is called a *data base* $\mathcal{D}$. Assuming that two distinct complete query answering functions exist (one for the ABox and one for the DBox), and defined a *knowledge base* as $\mathcal{KB} = \langle \mathcal{T}, \mathcal{W}, \mathcal{D} \rangle$, a uniform query function, based on the two distinct answering functions, can be implemented. No special capability is required from the DBox, except the one of (quickly) retrieving items or pairs of items satisfying requested conditions. These conditions are of the kind of *being in a class* – i.e., belonging to a unary table – or *being in relation with other items* – i.e., belonging to a binary table – and logical combinations of these, as it can be, for example, expressed in SQL.[10] Therefore, for the sake of simplicity, it will be assumed, here, that the DBox is implemented with a relational DBMS supporting SQL. Thus, in the following, tables and views are intended as usual in relational DBMS.

[7] D for data.

[8] $\mathcal{T}$ will be used to denote also the set of atomic terms appearing in $\mathcal{T}$, correctly classified in the taxonomy on the basis of their definitions.

[9] By $\mathcal{W}$ also the set of individuals described in $\mathcal{W}$ will be denoted.

[10] Of course, the external source of information, where the DBox searches for data, could be of any kind, provided only that it can be accessed via a sufficiently powerful query language (i.e., any query language equivalent to the portion of SQL used here), allowing, in this way, to manage also heterogeneous data sources.

4.2 Coupling

Coupling the terminology $\mathcal{T}$ with the data base $\mathcal{D}$ corresponds to associating some terms (concepts and roles) of $\mathcal{T}$ with tables or views in the DB representing $\mathcal{D}$. The coupling of $\mathcal{T}$ with $\mathcal{D}$ is performed in two steps. First, a partial mapping PM between primitive terms in $\mathcal{T}$ and the tables in DB must be given. Giving a mapping of a primitive term into a DB-table corresponds to giving its extension in the DB. Let the terms for which PM is defined be called $\mathcal{D}$-terms. Then, using PM, also non-primitive concepts can be recursively mapped into views of the DB. If the expanded definition[11] of a non-primitive concept contains both $\mathcal{D}$-terms and non-$\mathcal{D}$-terms, the view in which the concept is mapped does not contain all the instances of the concept. Therefore, non-primitive concepts with expanded definition containing both $\mathcal{D}$-term and non-$\mathcal{D}$-term cannot be completely managed in our system.

Thus, it must be assumed that the following conditions on $\mathcal{KB} = \langle \mathcal{T}, \mathcal{W}, \mathcal{D} \rangle$ are satisfied:

1. Every table in $\mathcal{D}$ must correspond to one primitive term in $\mathcal{T}$, called $\mathcal{D}$-term; $\mathcal{D}$-terms cannot be used in the expanded definition of any primitive term in $\mathcal{T}$.
2. The expanded definitions of non-primitive concepts in $\mathcal{T}$ must contain only $\mathcal{D}$-terms or no $\mathcal{D}$-term at all.

The aim of the constraint 1 is to avoid any need of consistency checking in case of conflicts between defining concepts and those used in the definitions. If, to ensure the avoidance of such conflicts, an exhaustive checking – that could involve also the extensional analysis of DB-tables – were provided, this constraint could be released.

All the information needed to correctly drive the query answering mechanism is the association of some terms with the corresponding tables or views in the DB. To this end, it is enough to know this association for primitive terms. Thus, only a partial mapping

$$PM : \mathcal{PT} \rightarrow DBtable$$

(where $\mathcal{PT}$ is the set of primitive terms in $\mathcal{T}$, and $DBtable$ the set of tables in the DB) must be given. The views corresponding to non-primitive concepts can be built via a recursive partial mapping

$$RM : \mathcal{T} \rightarrow DBtable \cup DBview$$

(where $DBview$ is the *virtual* set of views in the DB) that maps DL-expressions into corresponding SQL-expressions.

In the following, to simplify the description, it is assumed that concepts are mapped into unary tables with one column called `left`, and roles into binary tables with two columns called `left` and `right`. In the implemented system, of course, translation tables provide the substitution with the true names.

As an example, assume that non-primitively defined concepts that contain $\mathcal{D}$-terms in their expanded definition are constrained to use the sub-language with the only

[11]An *expanded* definition is a definition where each atom is replaced by its expanded definition recursively (if it has one). Of course, only acyclic definitions are allowed in $\mathcal{T}$.

`AND` and `SOME` operators; in this case RM can be defined as follows:[12]

```
RM((AND C D)) =
  SELECT DISTINCT  left
  FROM             RM(C), RM(D)
  WHERE            RM(C).left =
                    RM(D).left
```

if both $RM(C)$ and $RM(D)$ are defined;

```
RM((SOME R D)) =
  SELECT DISTINCT  left
  FROM             RM(R)
  WHERE            RM(R).right IN
                    RM(D)
```

if both RM(`R`) and $RM(D)$ are defined;

```
RM(T) =  PM(T)
```

if $PM(T)$ is defined;

and

```
RM(T) =  SELECT DISTINCT  *
         FROM             T_1
         UNION
         ⋮
         SELECT DISTINCT  *
         FROM             T_n
```

if $M(\mathtt{T}) = \{T_1, \ldots, T_n\}$, and $n > 0$.

Note that the last part of this definition (see below for the definition of M) takes into account all the tables and views corresponding to terms subsumed by `T`, whatever `T` is.

Of course RM could be extended to more general concepts, but in some cases the mapping would have to be carefully handled, due to the different semantics of DL and DB (e.g., for the `NOT` and the `ALL` operators).[13]

Note that, due to limitations of SQL in using sub-queries, the `SELECT` used in the definition of RM are not exactly legal, due to the recursive application of RM. This problem can be easily overcome if a `CREATE VIEW` corresponds to each application of RM, and the names of the corresponding views are placed in lieu of the recursive applications of RM.[14]

[12]Note that `R` stands for a role name, i.e., for an atomic role in $\mathcal{T}$, while C and D stand for concept names and expressions. In general, the `TYPEWRITER` font will be used for atomic terms.

[13]See footnote 5.

[14]Of course, this requires a pre-compilation of the DB with respect to the KB, but this is not a real overload of the presented query answering mechanism.

The function

$$M : \mathcal{T} \rightarrow 2^{DBtable \cup DBview}$$

used in the definition of RM returns the (possibly empty) set of tables/views necessary to retrieve all the instances (pairs) of a given concept (role) from the DB, that is:

$$M(\mathtt{T}) = \{RM(x) \mid \begin{array}{l} x \in subs(\mathtt{T}) \text{ and} \\ RM(x) \text{ is defined}\} \end{array}$$

where $subs(\mathtt{T})$ is the set of $\mathcal{T}$ terms classified under $\mathtt{T}$. Observe that RM and M are built starting from PM; thus, it is enough to include PM in the $\mathcal{KB}$ definition: $\mathcal{KB} = \langle \mathcal{T}, \mathcal{W}, \mathcal{D}, PM \rangle$.

4.3 Query Answering

The task of answering a query can now be described. Here, it is assumed that a *query* to $\mathcal{KB} = \langle \mathcal{T}, \mathcal{W}, \mathcal{D}, PM \rangle$ is an expression of the form $\lambda \overline{x}.(P_1 \wedge \ldots \wedge P_n)$, where $P_1, \ldots, P_n$ are atoms of the form $\mathtt{C}(x)$ or $\mathtt{R}(x, y)$, where $\mathtt{C}$ and $\mathtt{R}$ are concepts and roles in $\mathcal{T}$, and each of x and y appears in the tuple of variables $\overline{x} = \langle x_1, \ldots, x_m \rangle$ or is an individual constant in $\mathcal{W} \cup \mathcal{D}$. *Answering* a query to $\mathcal{KB}$ means finding a set $\{\overline{x}^1, \ldots, \overline{x}^m\}$ of tuples of individuals such that, for each tuple $\overline{x}^i$, $\lambda \overline{x}.(P_1 \wedge \ldots \wedge P_n)[\overline{x}^i]$ holds – explicitly or implicitly – in $\mathcal{KB}$. Such tuples will be called *answers* to the query and the set of all of them the *answer set*.

From the definition of answer to a query, it is obvious that, to avoid the generation of huge answer sets, free variables must not be used, that is, each variable appearing in $\overline{x}$ must appear also in the query body. Indeed, even stronger restrictions, that allow for a more efficient management of queries than the one here presented, are adopted (see [13]).

A query, to be answered, must be split into sub-queries that can be answered by the two specialized query answering functions of the DLMS and the DBMS. To this end, a *marking* of all the possible atomic sub-queries, corresponding to the terms in $\mathcal{T}$, is needed; a term P is said to be:

- KB-marked iff $RM(P)$ is undefined;
- Mixed-marked otherwise.

This marking reflect the fact that the instances (pairs of instances) of P are all in $\mathcal{W}$, or part in $\mathcal{W}$ and part in $\mathcal{D}$, respectively. The case of queries in which the atomic sub-queries correspond all to KB-marked terms is trivial (it is enough to submit it to the DLMS answering function). The case of queries with also Mixed-marked predicates is more difficult.

Let a generic query be written as

$$\lambda \overline{x}.(P_1^{KB} \wedge \ldots \wedge P_m^{KB} \wedge P_1^M \wedge \ldots \wedge P_n^M)$$

where the P_i^{KB} correspond to the KB-marked terms, and the P_i^M to the Mixed-marked terms. This query can be split in the two sub-queries

$$q^{KB} = \lambda \overline{x}_{KB}.(P_1^{KB} \wedge \ldots \wedge P_m^{KB})$$

and

$$q^M = \lambda \overline{x}_M.(P_1^M \wedge \ldots \wedge P_n^M).$$

As mentioned, q^{KB} can be straight answered by the DLMS. It will be shown below how it is possible to find an answer to q^M too.

4.3.1 Translating Queries into SQL

Because each predicate in the query

$$q^M = \lambda \overline{x}_M.(P_1^M \wedge \ldots \wedge P_n^M)$$

corresponds to a view in the DB – where the answers have to be searched in addition to those in the ABox – a translation of them into equivalent SQL queries must be provided. Of course, the views can easily be found via the recursive mapping RM. For each of the P_i^M in q^M the translation into an equivalent view is simply given by $RM(P_i^M)$. Thus, the SQL query corresponding to $q_i^M = \lambda \overline{y}_i.P_i^M$ – where $\overline{y}_i$ is the sub-tuple of $\overline{x}_M$ containing the only one or two variables used in P_i^M – is:

```
SELECT DISTINCT   select-body
FROM              RM(P_i^M)
WHERE             where-body
```

where the *select-body* contains $RM(P_i^M)$`.left`, $RM(P_i^M)$`.right`, or both, according to the fact that P_i^M is of the kind $\mathtt{C}(x)$ or $\mathtt{R}(x, a)$, $\mathtt{R}(a, y)$, or $\mathtt{R}(x, y)$, respectively – with x and y variables, and a constant. The `WHERE` clause is present only if $P_i^M = \mathtt{R}(x, a)$ or $P_i^M = \mathtt{R}(a, y)$; in these cases the *where-body* is

$$RM(\mathtt{R}).\mathtt{left} = a$$

or

$$RM(\mathtt{R}).\mathtt{right} = a$$

respectively.

In this way n partial answer sets (one for each P_i^M) are obtained. Of course, the queries have to be submitted also to the DLMS, in case also some individuals of $\mathcal{W}$ satisfy them.

Now, it is, ideally, enough to get the intersection of all the partial answer sets obtained by processing the sub-queries of q^M and q^{KB}, but, due to the scope of the variables of the queries, this cannot be efficiently performed in a direct way: a *merging* of the results is needed.

4.3.2 Merging the Results

A merging mechanism for partial answer sets must be provided. In fact, in each sub-query some of the variables in $\overline{x}$ may be unbound – that is, the proper tuple $\overline{y}$ of variables appearing in the sub-query may be a sub-tuple of $\overline{x}$. Therefore, the corresponding answer set has to be *completed*, that is, each unbound variable in $\overline{x}$ – i.e., those not appearing in $\overline{y}$ – must be made correspond to each instance in $\mathcal{KB}$, for all the found answers, considering all the possible combinations. However, in this way huge answer sets would be generated.

To solve this problem, a compact representation for the answer sets is needed. Let a generic partial answer set of a sub-query be written as $AS_{\overline{y}}$, where the variables of the original complete variable tuple $\overline{x}$ missing in $\overline{y}$ are $x_{p_1}, \ldots, x_{p_k}$. Its completion can be represented in a compact way as

$$AS_{\overline{x}} = \{\overline{I}^{\star} \mid \overline{I} \in AS_{\overline{y}}\}$$

where each $\overline{I}^{\star}$ is equal to $\overline{I}$ except that it is lengthened by filling the k missing positions $p_1, \ldots, p_k$ with any marker, e.g., a star '$\star$'. The star stands for any individual in $\mathcal{KB}$. Using this representation, the *merging* of answers sets can be efficiently managed, using the following `MERGE` algorithm.

```
MERGE($AS^1_{\overline{x}}, AS^2_{\overline{x}}, \ldots, AS^n_{\overline{x}}$) :
4.1 let result-list={$AS^1_{\overline{x}}, AS^2_{\overline{x}}, \ldots, AS^n_{\overline{x}}$};
4.2 choose two answer sets, $AS_1$ and $AS_2$, in result-list, where answers have
    at least one common position filled by individuals, i.e., not $\star$;[15]
4.3 merge $AS_1$ and $AS_2$ by collecting only those answers in $AS_1$ where each non-
    $\star$ filled position is filled by the same individual or by $\star$ in some answers in
    $AS_2$, and replace in the collected answers each $\star$ with the individuals in the
    corresponding position in all the matching answers of $AS_2$;
4.4 replace $AS_1$ and $AS_2$ in result-list with their merging computed in step
    4.3;
4.5 REPEAT from step 4.2 UNTIL only one item is left in result-list;
4.6 RETURN the only item left in result-list.
```

The steps to be performed to correctly answer a query can now be summarized as follows:

1 split the query as shown at the beginning of section 4.3 into q^{KB} and q^M;

2 submit q^{KB} to DLMS, and each of the atomic sub-queries $q^M_1, \ldots, q^M_n$ of q^M to the DBMS, after translation into SQL, and to the DLMS, as shown in section 4.3.1;

3 using the compact notation shown above, *complete* the answer sets $AS^{KB}_{\overline{x}_{KB}}$, $AS^{M_1}_{\overline{x}_{M_1}}, \ldots, AS^{M_n}_{\overline{x}_{M_n}}$ obtained with step 2, and generate $AS^{KB}_{\overline{x}}, AS^{M_1}_{\overline{x}}, \ldots, AS^{M_n}_{\overline{x}}$;

4 the overall answer set is

$$\texttt{MERGE}(AS^{KB}_{\overline{x}}, AS^{M_1}_{\overline{x}}, \ldots AS^{M_n}_{\overline{x}}).$$

5 Conclusions

The importance of an integrated view of description logics and databases has been shown, and our approach to deal with this topic has been presented. Within our technique, a third component – a DBox – can be added to the traditional TBox/ABox architecture of DLMS. By means of the DBox it is possible to couple the DLMS with several, possibly distributed and heterogeneous, source of data, and

[15]Such two sets always exist (see [13]).

to use all the systems for uniformly answering queries to knowledge bases realized with this extended paradigm.

In our first implementation of the system[16] the DLMS is LOOM [25], and the database query language is SQL, but also other systems could be easily used.

At present our tool is used in a natural language dialogue system prototype [12], whose domain and linguistic knowledge is represented in a LOOM KB and, for some large amount of raw data, in an INGRES DB. Currently, our system support a more expressive query language than the one previously presented: existentially quantified conjunctions of atomic formulæ can also be used. The study of the use of even more complex query-languages is planned, as well as a wider coverage of DL operators.

Acknowledgment

We would like to thank Enrico Franconi for the fruitful discussions and many hints about the content of this paper.

Our thanks are addressed also to Paul DeMaine, Chuck Karr, and the other organizers of KARP-95 [1] – where a short preliminary version of this paper has been presented – for their suggestion of submitting this paper for publication.

References

[1] *KARP-95 Second International Symposium on Knowledge Acquisition, Representation, and Processing*, Auburn, AL, September 1995.

[2] F. Baader, M. Buchheit, M. A. Jeusfeld, and W. Nutt, editors. *Working Notes of the KI'94 Workshop: KRDB'94 - Reasoning about Structured Objects: Knowledge Representation meets Databases*, number D-94-11 in DFKI Documents, Saarbrücken, Germany, September 1994. DFKI GmbH.

[3] F. Baader, M. Buchheit, M. A. Jeusfeld, and W. Nutt, editors. *KRDB'95: Reasoning about structured objects: Knowledge representation meets databases*, number D-95-12 in DFKI-Research-Report, Saarbrücken, Germany, 1995. DFKI GmbH.

[4] F. Baader, M. Buchheit, M. A. Jeusfeld, and W. Nutt, editors. *Working Notes of the ECAI-96 Workshop "Knowledge Representation Meets Databas (KRDB-96)"*, Budapest, August 1996.

[5] Clara Bagnasco, Paolo Bresciani, Bernardo Magnini, and Carlo Strapparava. Natural language interpretation for public administration database querying in the TAMIC demonstrator. In R.P. van de Riet, J.F.M. Burg, and A.J. van der Vos, editors, *Applications of Natural Language to Information Systems*, pages 163–174, Amsterdam, June 1996. IOS Press. Proceedings of the Second International Workshop.

[16]Indeed, the answering algorithm has been implemented in a more sophisticated way than the one presented in section 4.3, including also optimizations for reducing the number of accesses to the DB (see [13]).

[6] Domenico Beneventano, Sonia Bergamaschi, Stefano Lodi, and Claudio Sartori. Using subsumption in semantic query optimization. In A. Napoli, editor, *IJCAI Workshop on Object-Based Representation Systems*, pages 704–709, Chambery, France, August 1993.

[7] Sonia Bergamaschi and Claudio Sartori. On taxonomic reasoning in conceptual design. *ACM Transactions on Database Systems*, 17(3):385–422, September 1992.

[8] Alex Borgida and Ronald J. Brachman. Loading data into description reasoners. In *Proceeding of ACM SIGMOD '93*, 1993.

[9] Alexander Borgida. Knowledge representation, semantic modeling: Similarities and differnces. In Hannu Kangassalo, editor, *Entity-Relationship Approach: The Core of Conceptual Modelling*, pages 1–25. North-Holland, 1991. Proceedings of the Ninth International Conference on the Entity-Relationship Approach.

[10] R. J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[11] Ronald J. Brachman, Peter G. Selfridge, Loren G. Terveen, Boris Altman, Alex Borgida, Fern Halpern, Thomas Kirk, Alan Lazar, Deborah L. McGuinness, and Lori Alperin Resnick. Integrated support for data archaeology. In *ISMM International Conference on Information and Knowledge Management*, Baltimore, MA, November 1992.

[12] Paolo Bresciani. Use of loom for domain representation in a natural language dialogue system. Technical Report 9203-01, IRST, Povo, TN, Italy, March 1992. presented at LOOM Users Workshop, Los Angeles, March 23-24, 1992.

[13] Paolo Bresciani. Uniformly querying knowledge bases and data bases. In Baader et al. [2], pages 58–62.

[14] Paolo Bresciani. Querying databases from description logics. In Baader et al. [3], pages 1–4.

[15] Paolo Bresciani. Some research trends in knowledge representation and databases. In Baader et al. [4], pages 10–13.

[16] Paolo Bresciani and Enrico Franconi. Description logics for information access. In *Working Notes of the AI*IA 1996 Workshop on "Access, Extraction and Integration of Knowledge"*, Napoli, September 1996.

[17] Michael L. Brodie. Future Intelligent Information Systems: AI and database technologies working together. In *Readings in Artificial Intelligence and Databases*, chapter Epilogue, pages 623–641. Morgan Kaufman, San Matteo, California, 1988.

[18] Martin Buchheit, Manfred A. Jeusfeld, Werner Nutt, and Martin Staudt. Subsumption between queries to object-oriented databases. *Information Systems*, 19(1):33–54, 1994.

[19] *Proceedings of Second Conference on Information and Knowledge Management (CIKM '93)*, 1993.

[20] Premkumar T. Devanbu. Translating description logics to information server queries. In CIKM '93 [19].

[21] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In *Proc. of the 2 nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 151–162, Cambridge, MA, 1991.

[22] Chun-Nan Hsu and Craig A. Knoblock. Reformulating query plans for multidatabase systems. In CIKM '93 [19].

[23] Thomas Kirk, Alon Y. Levy, and Divesh Srivastava. The information manifold. In *1994 CAIA Workshop on Intelligent Access to Libraries*, 1994.

[24] Alon Y. Levy, Yehoshua Sagiv, and Divesh Srivastava. Toward efficient information gathering agents. In *Working Notes of the AAAI Spring Symposium "Software Agents"*, March 1994.

[25] R. MacGregor. Inside the LOOM description classifier. *SIGART Bulletin*, 2(3):88–92, 1991.

[26] Don P. McKay, Tim W. Finin, and Anthony O'Hare. The intelligent database interface. In *Proc. of AAAI-90*, pages 677–684, Boston, MA, 1990.

[27] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*, volume 422 of *Lecture Notes in Artificial Intelligence.* Springer-Verlag, Berlin, Heidelberg, New York, 1990.

[28] Jon A. Pastor, Donald P. McKay, and Timothy W. Finin. View-concepts: Knowledge-based access to databases. In *Proceedings of First Conference on Information and Knowledge Management (CIKM '92)*, Baltimore, 1992.

[29] Raymond Reiter. On closed-world data bases. In H. Gallaire and J. Minker, editors, *Logic nad Data Bases*, pages 55–76. Plenum Press, 1978.

[30] Albrecht Schmiedel. Semantic indexing based on description logics. In Baader et al. [2], pages 41–44.

[31] F. Tou, M. Williams, R. Fikes, A. Henderson, and T. Malone. Rabbit: An intelligent database assistant. In *Proc. AAAI'82*, 1982.